An Efficient and Simple Algorithm for Matrix Inversion

Ahmad Farooq1, Khan Hamid2

1King Khalid University, Abha, Kingdom of Saudi Arabia

(drfarooqa@gmail.com)

2National University of Computer and Emerging Sciences (NUCES), Pakistan

**0. Abstract**

In this paper a new algorithm is proposed for finding inverse and determinant of a given matrix in one go. The algorithm is straightforward in understanding and manual calculations. Computer implementation of the algorithm is extremely simple and is quite efficient in time and memory utilization. The algorithm is supported by an example. The number of multiplication/division performed by the algorithm is exactly however; its efficiency lies in the simplicity of coding and minimal utilization of memory. Simple applicability and reduced execution time of the method is validated form the numerical experiments performed on test problems. The algorithm is applicable in the cases of pseudo inverses for non-square matrices and solution of system of linear equations with minor modification.

**Keywords**: Algorithm, Pseudo Inverse, Matrix Inversion, Linear Algebra

**Acknowledgement**: The authors are indebted to the review, comments and Lemma suggested by Prof. Toru Nakahara of Saga University, Japan.

1. **Introduction**

The problem of finding inverse is one of the important problems in applied sciences and engineering. There are several methods available for finding inverses and for solving system of linear equations like iterative methods, Gauss elimination procedure and decomposition methods [Burden 2001, Fill 97, Najafi 2006, Rao 1971]. Recently many researchers worked on the area of matrix inversion [Chang 2006, Mikkawy 2006, Vajargah 2007]. If inverse of coefficient matrix *A* in a given linear system is known then the solution can be found by. In fact, the inversion of matrix is more generic requirement than the solution of linear system of equations in numerical linear algebra. Many algorithms exist to obtain numerical inversion for the given nonsingular matrix. A survey of these algorithms shows that efficiency, accuracy and simplicity of these algorithms can still be improved. However, most of these algorithms focus on particular types of matrices such as positive definite, diagonally dominant, banded and symmetric matrices etc.

In this work a new algorithm is developed for finding inverse of a given matrix. This approach is simpler and efficient than the exiting techniques and is applicable in general irrespective of the structure of the matrix. The manual calculations are straightforward and computer implementation is extremely easy. The memory utilization is minimal i.e., it stores only the original matrix and replaces it gradually by the inverse. The most important and unique features of the algorithm is the ability of finding inversion and determinant in one go.

The rest of the paper is organized as follows. In section 2 we are presenting the simple algorithm. To emphasize the simplicity of computer implementation the code is also given in this section. Section 3 demonstrates the use of the algorithm through a numerical example. Section 4 is devoted to comparing computational complexity of the algorithm. In section 5 the performance results of the technique applied to the inversion of various matrices is given. Finally some conclusions are given in section 6.

**2 (a). The Simple Algorithm for matrix inversion**

The algorithm assumes to take a square matrix of dimension *n*. The inverse is calculated in *n* iterations. In each iteration *p,* all the existing elements of *A* change to new values After the last iteration i.e. when , will be the elements of the inverse. The determinant of the matrix (denoted by *d*)is also calculated iteratively through successive multiplication of the pivot selected in each iteration. In this algorithm the pivots are selected diagonally starting from to If any pivot is found to be zero i.e., then inverse cannot be calculated. If an inverse is calculated then *d* will contain the determinant of *A.*

A simple improvement to the algorithm is to go to the next diagonal element (in case of zero pivot) and revisit the zero diagonal element later. Probably by that time it would become non zero. Note that in step 7 of the following algorithm  on the LHS means that the latest value of the pivot row is to be used in the calculations.

Step 1: Let *p =* 0*, d =* 1*;*

Step 2:

Step 3: If then cannot calculate inverse, go to step 10.

Step 4: 

Step 5: Calculate the new elements of the pivot row by:



Step 6: Calculate the new elements of the pivot column by:



Step 7: Calculate the rest of the new elements by:



Step 8: Calculate the new value of the current pivot location:



Step 9: If *p < n* go to step 2 (*n* the dimension of the matrix *A*).

Step 10: Stop. If inverse exists- *A* contains the inverse and *d* is the determinant.

**Lemma:**

Let *A* be a non singular matrix of size 2 and M is obtained from *A* through acting suitable matrix transformations. Whereas, M= *A* is obtained from M through acting and inserting suitable matrix transformation, then

(1) F the matrix obtained through the 1st iteration of the algorithm coincides with M.

(2) F(=*A*) the matrix obtained through the 2nd iteration of the algorithm, coincides with M.

Proof: 

Let , let , let 









This gives



by solving 1 and 2 we get

,

by putting the value of *x* and *y* in equation  and multiplying with  we get  so



let 

let  let , let 















let  & 



 , 





**2 (b). The Code (in C language)**

In the following lines the above algorithm is shown implemented in C language. The objective of giving the code here is mainly to demonstrate the simplicity of the algorithm. The code could be further optimized for performance in terms of time and memory in various ways. *MatB* in the code represent the original matrix provided to the function which will iteratively convert to the inverse. The dimension of the matrix is represented by *size*. Whereas *MAX* is assumed to be a *#def* constant this should be assigned a value equal to the maximum possible dimension of the matrix. If desired *MAX* could be replaced by appropriate numeric value. The function will return a float value. A nonzero return-value represent the determinant of the matrix and a zero return value indicates that the inverse could not be calculated and *MatB* does not contain the inverse.

float inverseMat(float MatB[MAX][MAX], int size)

{

float pivot, det=1.0;

int i, j, p;

for(p=1; p <= size; p++)

{

pivot = MatB[p][p];

det= det \* pivot;

if (fabs(pivot) < 1e-5) return 0;

for (i = 1; i<= size; i++)

MatB[i][p] = - MatB[i][p] / pivot;

for (i = 1; i<= size; i++)

if (i != p)

for (j= 1; j <= size; j++)

if (j != p)

MatB[i][j] = MatB[i][j] + MatB[p][j] \* MatB[i][p];

for (j= 1; j <= size; j++)

MatB[p][j] = MatB[p][j]/ pivot;

MatB[p][p] = 1/ pivot;

}

return det;

}

**3. Numerical Example**

Let assume that we want to calculate the inverse of matrix *A*.



The dimension of the matrix is 3. Therefore, inverse will be calculated in 3 iterations. In iteration 1 we select  as pivot (step 3). Therefore row 1 and column 1 are the pivot row and column.







**Iteration 1:**

The new pivot row values (except the pivot) are calculated using step 5.

The new pivot column values (except the pivot) are calculated using step 6.

The values of the elements excluding pivot row and pivot column are calculated using step 7.

The new value of the pivot is calculated using step 8. The current value of the determinant is 2 i.e., the pivot of first iteration (as per step 4).







**Iteration 2:**

In iteration 2 we select  as pivot (step 3). Therefore, row 2 is the pivot row and column 2 is the pivot column.

The new pivot row values (except the pivot) are calculated using step 5

The new pivot column values (except the pivot) are calculated using step 6.



The values of the elements excluding pivot row and pivot column are calculated using step 7.

The new value of the pivot is calculated using step 8. The partial value of the determinant so far is  (as per step 4).



**Iteration 3:**

In iteration 3 we select  as pivot (step 3). Therefore, row 3 is the pivot row and column 3 is the pivot column.

The new pivot row values (except the pivot) are calculated using step 5

The new pivot column values (except the pivot) are calculated using step 6.

The values of the elements excluding pivot row and pivot column are calculated using step 7.

The new value of the pivot is calculated using step 8. The current value of the determinant is  (as per step 4).

At the end of third iteration we obtain the inverse of the matrix. The determinant of the matrix is  (as per step 4).



**4. Analysis of calculations performed**

Observing the simple algorithm given in 2(a) the number of maximum arithmetic operations required for inversion of an *n* by *n* matrix can be calculated quite easily.

Number of multiplications/division:

divisions for pivot row,

divisions for pivot column,

 multiplications for rest of the elements and

1 division of pivot.

All these multiplications/divisions to be performed *n* times.

Therefore, total multiplications/divisions are .

Number of addition/subtraction:

0 for pivot row,

0 for pivot column,

0 for pivot and

 additions for rest of the elements.

All these addition/subtraction to be performed *n* times.

Therefore, total summations are .

Determinant requires  multiplications.

**5. Comparative results of computer simulation**

The algorithm was tested for comparison of operational timing estimates with Gauss Jordan method. The simulations were performed on portable laptop with following specifications:

• Operating System: Microsoft Windows XP Professional (5.1, Build 2600)

• Processor: Intel(R) Pentium(R) M Processor 1.73 GHz

• Memory: 512 MB DDR RAM

Size of Matrix Gauss Jordan New Technique

30\*30 0.016 0

50\*50 0.032 0

100\*100 0.26 0.015

150\*150 0.54 0.047

200\*200 1.13 0.125

250\*250 2.05 0.234

**6. Conclusions**

This work has presented an algorithm for matrix inversion. The algorithm is straightforward for manual calculations and extremely simple for computer programming. It can calculate the inverse and the determinant in one go and actually requires only *n-1* extra multiplication for the calculation of determinant. There are indications that the algorithm can be used with improvements in variety of situation e.g., solution of simultaneous equations, calculations of determinants only, calculating error bounds on the elements of the inverse etc.

The algorithms requires exactlymultiplications/divisions to calculate the inverse, however, its efficiency lies in the minimal utilization of memory and very simple computer coding, requiring only basic arithmetic operation.

**References**

Burden, R.L. and Fairs, J.D.(2001), ‘Numerical Analysis’, 7th ed., Brooks/Cole, 2001.

Chang, F. C. (2006), ‘Inverse of a perturbed matrix’, Applied Mathematical Letters 19 (2006) 169-173.

Fill, J. and Fishkind, D.E. (1997), ‘Moore-Penrose generalized inverse for sums of matrix’, Annual Mathematical Statistics 18 (1997)

Mikkawy, M. E. and Karawia, A. (2006), ‘Inversion of general tridiagonal matrices’, Applied Mathematical Letters. 19 (2006), pp. 712-720.

Najafi, H. S. and Solary, M. S. (2006), ‘Computational algorithms for computing the inverse of a square matrix, quasi-inverse of a non-square matrix and block matrices’, Applied Mathematical Computation, 183 (2006), pp. 539-550.

Rao, C. and Mitra, S. K. (1971), ‘Generalized inverse of matrices and its applications’, Wiley, New York, 1971.

Vajargah, B. F. (2007), ‘New advantage to obtain accurate matrix inversion’. Applied Mathematical Computation, doi 10.1016/j.amc.2006.12.060.